Product Requirements Document (PRD)

Cloud SSI Wallet — Frontend Web Client

Version: 1.0
Date: February 20, 2026
Status: Draft
Author: Product Team

---

1. Executive Summary

This document specifies the product requirements for building a **responsive web-based frontend client** for an existing **cloud Self-Sovereign Identity (SSI) wallet backend**. The wallet enables users to receive, store, visualize, and present **Verifiable Credentials (VCs)** — specifically in the **mDoc format** (ISO/IEC 18013-5) — through standard protocols: **OpenID for Verifiable Credential Issuance (OpenID4VCI 1.0)** and **OpenID for Verifiable Presentations (OpenID4VP 1.0)**.[^1][^2]

The frontend is a thin client that communicates exclusively with the backend API. All cryptographic operations, credential storage, and protocol handling happen server-side. The frontend's responsibility is to provide an intuitive, mobile-first user interface for interacting with the wallet's capabilities.[^3][^4]

---

2. Background & Domain Context

This section is essential for the developer. It explains the domain concepts needed to understand what this application does.

2.1 What Is Self-Sovereign Identity (SSI)?

SSI is a model where individuals (or organizations) control their own digital identity without relying on a central authority. In SSI, there are three roles:[^5]

- **Issuer**: An entity that creates and signs a digital credential (e.g., a government issuing a Photo ID, an airport issuing a boarding pass).
- **Holder (Wallet)**: The entity that receives, stores, and presents credentials. This is what we are building the frontend for.

- **Verifier**: An entity that requests and validates credentials from the holder (e.g., an airline gate checking your Photo ID).

The credential lifecycle is: Issuer → issues credential to → Holder/Wallet → presents credential to → Verifier.

2.2 What Are Verifiable Credentials (VCs)?

A Verifiable Credential is a tamper-evident digital document that contains claims about a subject (e.g., "John Doe has a valid Photo ID") and is cryptographically signed by the issuer. The holder stores VCs in their wallet and can selectively present them to verifiers.[^6][^7]

2.3 What Is mDoc (ISO 18013-5)?

mDoc is a credential format defined by **ISO/IEC 18013-5**, originally designed for mobile driving licenses. It is a binary format (CBOR-encoded), not JSON. Each mDoc has:[^8][^7][^6]

- A **doc type** identifier (e.g., `org.iso.18013.5.1.mDL` for driving licenses)
- **Namespaces** containing key-value data fields (e.g., `family_name`, `given_name`, `portrait`)
- A **certificate chain** linking the credential back to a trusted Certificate Authority (CA)
- Support for **selective disclosure** — the holder can share only specific fields

mDoc credentials require an **X.509 certificate chain** for trust validation (unlike DID-based VCs which use decentralized identifiers). This means the issuer must have a certificate signed by a CA, and the verifier must have the CA's root certificate in its trust anchor configuration.[^9][^10]

2.4 What Are OpenID4VCI and OpenID4VP?

These are the two protocols the wallet uses to interact with issuers and verifiers:[^2][^11][^1]

- **OpenID4VCI (OpenID for Verifiable Credential Issuance)**: Protocol for receiving credentials. An issuer generates a **credential offer URI** (starting with `openid-credential-offer://...`), which the wallet consumes to retrieve and store the credential.
- **OpenID4VP (OpenID for Verifiable Presentations)**: Protocol for presenting credentials. A verifier generates an **authorization request URI** (starting with `openid4vp://authorize?...`), which the wallet processes to share selected credential data.

2.5 What Is the Neoke ID Node?

The backend is a **Neoke ID Node** — a server that exposes an HTTP API for identity operations. It handles authentication, credential management, cryptographic key management, and the

OpenID4VCI/OpenID4VP protocol flows. The frontend communicates exclusively with this API.[^4][^9]

The specific ID Node instance for this project:

Property	Value
Host	`b2b-poc.id-node.neoke.com`
DID	`did:web:b2b-poc.id-node.neoke.com`
API Key	`dk_Uz4O7Vty17NZot4hdu5RIegRJnQUkeF3nmjNnXGbSOE`

[^3][^4]

---

3. Goals & Objectives

3.1 Primary Goal

Build a **responsive web application** that serves as the frontend for the cloud wallet, allowing users to perform all wallet operations through a graphical interface instead of API calls or command-line tools.[^3]

3.2 Success Criteria

- A non-technical user can receive, view, and present credentials without writing any code
- The interface works on mobile devices (phone-first) and desktops
- All three core wallet functions are supported: receive credentials, present credentials, manage consents
- The UX resembles Apple Wallet's card stacking paradigm[^3]

---

4. User Personas

4.1 Primary Persona: PoC Demo User

- **Who**: A project stakeholder or partner testing the SSI credential flow during a proof-of-concept demonstration (e.g., the Doha–Hong Kong travel route PoC)[^10]
- **Tech level**: Understands the concept of digital identity but does not write code

- **Goal**: Receive a Photo ID credential from an issuer, view it, and present it to a verifier — all from a web browser on their phone or laptop
- **Context**: Will receive credential offer URIs or QR codes from issuers, and authorization request URIs or QR codes from verifiers[^10]

4.2 Secondary Persona: Developer / Integrator

- **Who**: A developer integrating with the wallet for testing
- **Tech level**: Comfortable pasting URIs manually
- **Goal**: Quickly test issuance and presentation flows without using curl/Postman

---

5. Core Features & Functional Requirements

5.1 Feature 1: Authentication & Session Management

Description: The frontend must authenticate with the ID Node backend to obtain a bearer token, and manage the session lifecycle.

Requirements:

ID	Requirement	Priority
AUTH-01	On app launch, display a login screen requesting the user's API Key	Must
AUTH-02	On submission, call `POST /:/auth/authn` with header `Authorization: ApiKey <key>` to obtain a bearer token	Must
AUTH-03	Store the returned `token` and `expiresAt` in memory (not localStorage, for security)	Must
AUTH-04	Include the bearer token as `Authorization: Bearer <token>` in all subsequent API requests	Must

ID	Requirement	Priority
AUTH-05	Monitor token expiry. Show a re-authentication prompt when the token expires (1-hour TTL) or when any API call returns 401	Must
AUTH-06	Provide a "Logout" action that clears the token and returns to the login screen	Must
AUTH-07	After successful authentication, display the node's DID and host as confirmation of connection	Should

API Reference:[^4]

```
None
POST https://b2b-poc.id-node.neoke.com/:/auth/authn
Header: Authorization: ApiKey dk_Uz4O7Vty17NZot4hdu5RIegRJnQUkeF3nmjNnXGbSOE
Response: { "token": "eyJhbG...", "expiresAt": 1740000000000 }
```

5.2 Feature 2: Credential Dashboard (Home Screen)

Description: The main screen of the wallet displaying all stored credentials in a visual, card-based layout inspired by Apple Wallet.[^3]

Requirements:

ID	Requirement	Priority
DASH-01	On load, fetch all stored credentials from `GET /:/credentials`	Must
DASH-02	Display credentials as **stacked cards**, visually	Must

ID	Requirement	Priority
	overlapping like Apple Wallet. Only the top portion (header) of each card should be visible in the stack, showing: credential type, issuer name	
DASH-03	Cards should use the credential's **cover/display metadata** (background color, text color, logo) if available in the credential data. Fall back to a default card design if metadata is absent	Must
DASH-04	Support **dynamic credential types** — the UI must not hardcode any specific credential schema. Field names and values come from the credential data	Must
DASH-05	Display an empty state message ("No credentials stored yet") with a CTA to receive one when no credentials are stored	Must
DASH-06	Support pull-to-refresh (mobile) or a refresh button (desktop) to re-fetch credentials	Should
DASH-07	Cards should render correctly for all supported credential formats: mDoc, JWT-VC (if present)	Must

API Reference:[^4]

```
None
GET https://b2b-poc.id-node.neoke.com/:/credentials
Header: Authorization: Bearer <token>
Response: [{ "id": "urn:uuid:...", "type": ["VerifiableCredential"], ... }]
```

UX Specification:[^3]

- Apple Wallet–style stacking: cards overlap vertically, showing ~60px of each card's header
- The most recently added credential should be at the top of the stack
- Smooth animation when scrolling through cards

5.3 Feature 3: Credential Detail View

Description: When a user taps/clicks on a credential card, it should expand to show the full credential details.[^3]

Requirements:

ID	Requirement	Priority
DETAIL-01	Tapping a card animates it to expand to full screen (or a modal/slide-up panel on mobile)	Must
DETAIL-02	Display **all credential fields dynamically**. The fields are not predetermined — they are provided by the credential schema. Iterate over the credential's claims/attributes and render each as a labeled key-value pair	Must
DETAIL-03	For mDoc credentials, parse the namespace structure and group fields by namespace (e.g., `org.iso.18013.5.1`)	Must

ID	Requirement	Priority
DETAIL-04	Render the following field types appropriately: strings (as text), dates (formatted), images/portraits (as inline image from base64 or URI), booleans (as Yes/No or checkmark)	Must
DETAIL-05	Display credential metadata: credential ID, issuer DID, issuance date, expiration date, credential status (active/suspended/revoked)	Must
DETAIL-06	Provide a "Back" or close gesture to return to the stacked card view	Must
DETAIL-07	If the credential has cover/display configuration, render it as the card header with logo and colors	Should

UX Notes:[^3]

- The main card stack screen should show only generic info (type + issuer)
- The expanded detail view shows everything
- Fields should be rendered dynamically — the developer must NOT hardcode field layouts for specific credential types

---

5.4 Feature 4: Receive Credential (OpenID4VCI Flow)

Description: Allow the user to receive a new credential from an issuer using the OpenID4VCI protocol.[^2][^4][^3]

User Flow:

1. User obtains a credential offer URI from an issuer (either by scanning a QR code or by pasting the URI)

2. The frontend sends the URI to the backend
3. The backend processes the OID4VCI flow and returns the credential
4. The frontend displays a **consent screen** showing what is being issued and by whom
5. Upon user approval, the credential is stored and appears in the dashboard

Requirements:

ID	Requirement	Priority
RECV-01	Provide a **QR code scanner** (using device camera) that can read QR codes containing `openid-credential-offer://...` URIs	Must
RECV-02	Provide a **text input field** where users can paste an offer URI manually	Must
RECV-03	The QR scanner and text input should be accessible from the same screen/action point (e.g., a floating action button or tab)	Must
RECV-04	Detect the URI scheme: if it starts with `openid-credential-offer://`, route to the receive flow. If it starts with `openid4vp://authorize?`, route to the present flow	Must
RECV-05	Call `POST /:/oid4vci/receive` with the offer URI and a key ID	Must
RECV-06	Before finalizing, display a **consent screen** showing: (a) what credential is being issued (type, fields); (b) who is issuing it (issuer	Must

ID	Requirement	Priority
	identifier/name); (c) the data that will be included	
RECV-07	The consent screen must have explicit "Accept" and "Decline" buttons	Must
RECV-08	If the issuance flow requires a **Transaction PIN**, display a PIN input field for the user to enter it	Should
RECV-09	If the issuance is **deferred** (issuer reviews before delivering), show a pending state and allow the user to check back later	Should
RECV-10	On success, show a confirmation animation/message and add the credential to the dashboard	Must
RECV-11	On error, show a clear, human-readable error message	Must

API Reference:[^4]

```
None
POST https://b2b-poc.id-node.neoke.com/:/oid4vci/receive
Header: Authorization: Bearer <token>
Body: { "offer_uri": "openid-credential-offer://...", "keyId": "<key-id>" }
Response: { "credential": { "id": "...", "type": [...], ... } }
```

Consent Screen UX Specification:[^12][^3]

- Must display the data dynamically (fields come from the issuance request, not hardcoded)

- Show the issuer's identity prominently (DID, name if resolvable)
- Use a clear visual hierarchy: issuer at top, credential type, then a list of claims/fields being issued
- Consent is a GDPR-relevant action — it must be explicit and informed[^12]

5.5 Feature 5: Present Credential (OpenID4VP Flow)

Description: Allow the user to respond to a verifier's request by presenting credential data using the OpenID4VP protocol.[^11][^4][^3]

User Flow:

1. User obtains a VP request URI from a verifier (QR code or pasted URI)
2. The frontend sends the URI to the backend for **preview**
3. The backend returns what the verifier is requesting, which credentials match, and who the verifier is
4. The frontend displays a **consent screen** showing: who is requesting, what data they want, and for what purpose
5. The user selects which credential(s) to share and approves
6. The frontend calls the respond endpoint to execute the presentation
7. The frontend shows the result (success or failure)

Requirements:

ID	Requirement	Priority
PRES-01	Accept VP request URIs (`openid4vp://authorize?...`) via QR scanner or text input (same UI as Feature 4)	Must
PRES-02	Call `POST /:/auth/siop/respond/preview` with the request URI to get preview information	Must
PRES-03	Display a **presentation consent screen** showing: (a) the verifier's identity (client ID, name if available); (b) the specific fields/claims being	Must

ID	Requirement	Priority
	requested; (c) which stored credentials match the request; (d) the reason/purpose for the request (if provided)	
PRES-04	If multiple credentials match, allow the user to select which one to present	Must
PRES-05	Provide explicit "Share" and "Decline" buttons on the consent screen	Must
PRES-06	On approval, call `POST /:/auth/siop/respond` with the request URI	Must
PRES-07	Display the response status (success/redirect URI, or error details)	Must
PRES-08	If the request includes **transaction data** (consent text the holder must acknowledge), display it prominently before the share button	Should
PRES-09	Visually indicate which specific fields from the credential will be shared (selective disclosure) — highlight the requested fields vs. the full credential	Should

API Reference:[^4]

Step 1 — Preview:

```
None
POST https://b2b-poc.id-node.neoke.com/:/auth/siop/respond/preview
Header: Authorization: Bearer <token>
Body: { "request": "openid4vp://authorize?..." }
Response: { "verifier": { "clientId": "..." }, "queries": [...],
"matchedCredentials": [...] }
```

Step 2 — Respond:

```
None
POST https://b2b-poc.id-node.neoke.com/:/auth/siop/respond
Header: Authorization: Bearer <token>
Body: { "request": "openid4vp://authorize?..." }
Response: { "status": "ok", "redirectUri": "..." }
```

Consent Screen UX Specification:[^12][^3]

- Verifier identity at top (who is asking)
- Requested fields listed clearly (what they want)
- Purpose statement if available (why they need it)
- Matched credential shown as a card preview
- The user must be able to review exactly which data will be shared before confirming
- This is a privacy-critical action — design must inspire confidence and transparency

---

5.6 Feature 6: QR Code Scanner

Description: An integrated camera-based QR code reader to ingest credential offer URIs and VP request URIs.[^3]

Requirements:

ID	Requirement	Priority
QR-01	Access the device camera via the browser's MediaDevices API (`getUserMedia`)	Must
QR-02	Decode QR codes in real-time using a JavaScript	Must

ID	Requirement	Priority
	QR decoding library (e.g., `jsQR`, `zxing-js`, or `html5-qrcode`)	
QR-03	On successful scan, auto-detect the URI scheme and route to the appropriate flow (receive or present)	Must
QR-04	Show a viewfinder overlay on the camera feed	Should
QR-05	Provide a fallback for devices without cameras: show the text input field instead	Must
QR-06	Handle camera permission denial gracefully with a clear message	Must

---

6. Non-Functional Requirements

6.1 Responsive Design

ID	Requirement	Priority
NFR-01	**Mobile-first** design: the primary target is a smartphone browser. All interactions must be thumb-friendly	Must
NFR-02	Responsive breakpoints: Mobile (< 640px), Tablet (640–1024px), Desktop (> 1024px)	Must

ID	Requirement	Priority
NFR-03	On mobile: full-width cards, bottom-sheet modals, large touch targets (min 44×44px)	Must
NFR-04	On desktop: centered content area (max-width ~480px, simulating a phone-like experience, or a wider layout if preferred)	Should
NFR-05	All text must be legible at default zoom level. Minimum font size: 14px body, 12px labels	Must

6.2 Performance

ID	Requirement	Priority
NFR-06	Initial page load under 3 seconds on a 4G connection	Should
NFR-07	API calls must show loading indicators (spinner, skeleton screens)	Must
NFR-08	Smooth animations at 60fps for card stacking/expanding interactions	Should

6.3 Security

ID	Requirement	Priority
NFR-09	Bearer tokens must never be persisted in `localStorage` or `sessionStorage`. Use in-memory storage only. If the tab closes, the user must re-authenticate	Must
NFR-10	All API calls must use HTTPS	Must

ID	Requirement	Priority
NFR-11	The API key should be masked (input type="password") on the login screen	Must
NFR-12	Implement CORS-aware requests (the backend must also be configured for CORS)	Must

6.4 Accessibility

ID	Requirement	Priority
NFR-13	Semantic HTML structure (proper headings, landmarks, button elements)	Should
NFR-14	Keyboard navigable for all actions	Should
NFR-15	Sufficient color contrast (WCAG AA minimum)	Should
NFR-16	Screen reader–compatible labels on all interactive elements	Should

6.5 Browser Compatibility

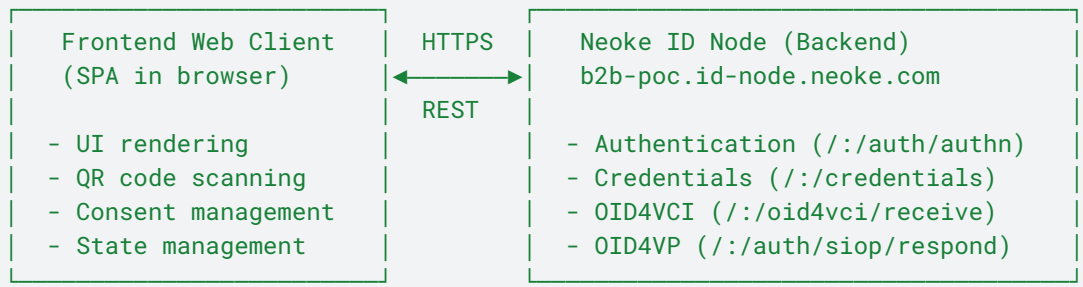
ID	Requirement	Priority
NFR-17	Support latest 2 versions of: Chrome, Safari (iOS + macOS), Firefox, Edge	Must
NFR-18	Camera access for QR scanning requires HTTPS and modern browser APIs	Must

---

7. Technical Architecture

7.1 Architecture Overview

None



The frontend is a **Single Page Application (SPA)** that communicates with the Neoke ID Node backend API via REST over HTTPS. All business logic, cryptographic operations, and protocol handling occur on the backend.[^4]

7.2 Recommended Technology Stack

The developer may choose their preferred stack, but here is a recommended setup:

Layer	Recommendation	Rationale
Framework	React (with Vite) or Next.js (static export)	Wide ecosystem, component-based architecture
Styling	Tailwind CSS	Utility-first, responsive by default, fast prototyping
State Management	React Context + useReducer, or Zustand	Lightweight, suitable for the app's scope
QR Scanner	`html5-qrcode` or `@zxing/browser`	Well-maintained browser QR libraries
HTTP Client	`fetch` API or Axios	Native or lightweight HTTP calls

Layer	Recommendation	Rationale
Animation	Framer Motion or CSS transitions	Smooth card stack animations
Language	TypeScript	Type safety for API contracts
Deployment	Static hosting (Vercel, Netlify, or S3 + CloudFront)	No server needed — it's a client-side SPA

7.3 API Endpoint Summary

Endpoint	Method	Purpose	Auth Required
`/:/auth/authn`	POST	Get bearer token from API key	No (API key in header)
`/:/credentials`	GET	List all stored credentials	Yes
`/:/oid4vci/receive`	POST	Receive a credential via OID4VCI offer	Yes
`/:/auth/siop/respond/preview`	POST	Preview a VP request (what the verifier wants)	Yes
`/:/auth/siop/respond`	POST	Respond to a VP request (present credentials)	Yes

[^4]

7.4 API Request/Response Contracts

Authentication:

```JSON
// Request
POST /:/auth/authn
Headers: { "Authorization": "ApiKey
dk_Uz4O7Vty17NZot4hdu5RIegRJnQUkeF3nmjNnXGbSOE" }
```

```
// Response
{ "token": "eyJhbG...", "expiresAt": 1740000000000 }
```

List Credentials:

```
JSON
// Request
GET /:/credentials
Headers: { "Authorization": "Bearer <token>" }

// Response (array of credential objects)
[
  {
    "id": "urn:uuid:...",
    "type": ["VerifiableCredential", "PhotoID"],
    "issuer": "did:web:issuer.id-node.neoke.com",
    "issuanceDate": "2026-02-20T10:00:00Z",
    "expirationDate": "2027-02-20T10:00:00Z",
    "credentialSubject": { ... },
    // mDoc-specific: docType, namespaces, etc.
  }
]
```

Receive Credential:

```
JSON
// Request
POST /:/oid4vci/receive
Headers: { "Authorization": "Bearer <token>", "Content-Type":
"application/json" }
Body: { "offer_uri": "openid-credential-offer://...", "keyId":
"62d48bb1-738b-491f-8af7-a645e51f908a" }

// Response
{ "credential": { "id": "...", "type": [...], ... } }
```

Preview VP Request:

```
JSON
// Request
POST /:/auth/siop/respond/preview
Headers: { "Authorization": "Bearer <token>", "Content-Type":
"application/json" }
Body: { "request": "openid4vp://authorize?..." }

// Response
{
  "verifier": { "clientId": "did:web:verifier.example.com" },
  "queries": [
    { "docType": "org.iso.18013.5.1.mDL", "requestedFields": ["family_name",
"given_name", "portrait"] }
  ],
  "matchedCredentials": [
    { "id": "urn:uuid:...", "type": [...], ... }
  ]
}
```

Respond to VP Request:

```
JSON
// Request
POST /:/auth/siop/respond
Headers: { "Authorization": "Bearer <token>", "Content-Type":
"application/json" }
Body: { "request": "openid4vp://authorize?..." }

// Response
{ "status": "ok", "redirectUri": "https://verifier.example.com/callback?..." }
```

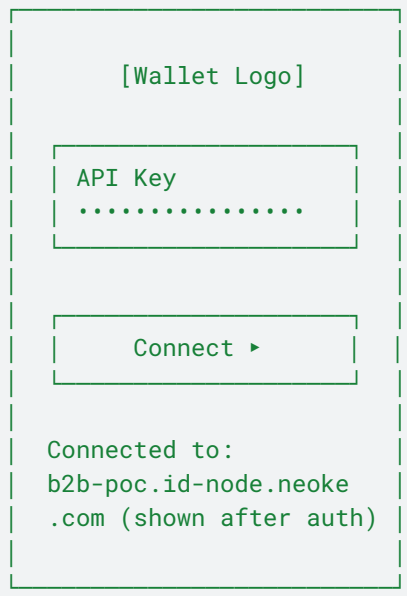
> **Developer Note**: The exact response shapes above are representative. The developer should make an initial call to each endpoint and inspect the actual response structure, then build TypeScript interfaces accordingly. The frontend must handle any credential schema dynamically — do not hardcode field names.

---

8. User Interface Wireframes (Descriptive)

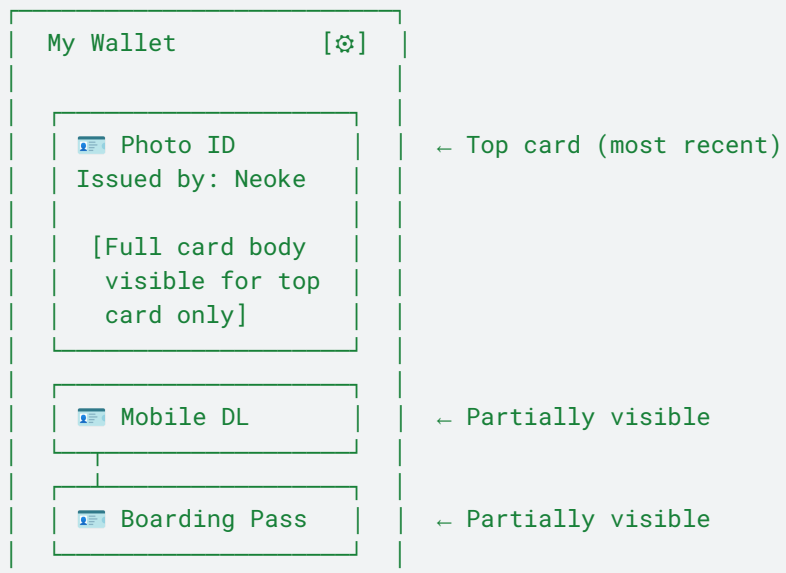
8.1 Screen: Login

None



8.2 Screen: Credential Dashboard (Card Stack)

None



```
|                              |
|         [📷 Scan]            |  ← FAB or bottom bar
|  [paste URI here...    ]     |  ← Text input
└──────────────────────────────┘
```

8.3 Screen: Credential Detail (Expanded Card)

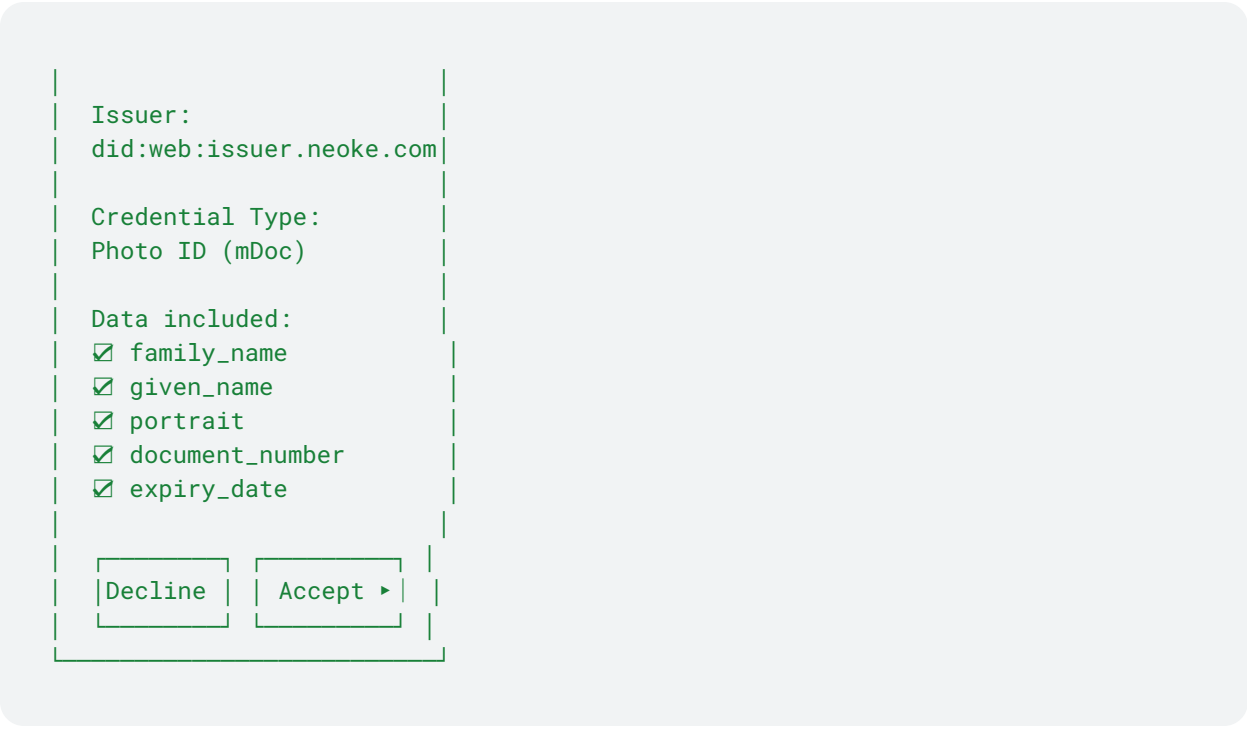
None

```
┌──────────────────────────────┐
|  ← Back                      |
|                              |
|   ┌──────────────────────┐   |
|   |                      |   |
|   |   [Credential        |   |
|   |    Card Header       |   |
|   |    with Logo]        |   |
|   |                      |   |
|   └──────────────────────┘   |
|                              |
|  Type: Photo ID              |
|  Issuer: did:web:...         |
|  Issued: 2026-02-20          |
|  Expires: 2027-02-20         |
|  Status: ● Active            |
|                              |
|  ── Fields ──                |
|  Family Name: Doe            |
|  Given Name: John            |
|  Document #: ABC123          |
|  Portrait: [image]           |
|  ... (all fields)            |
|                              |
└──────────────────────────────┘
```

8.4 Screen: Receive Consent

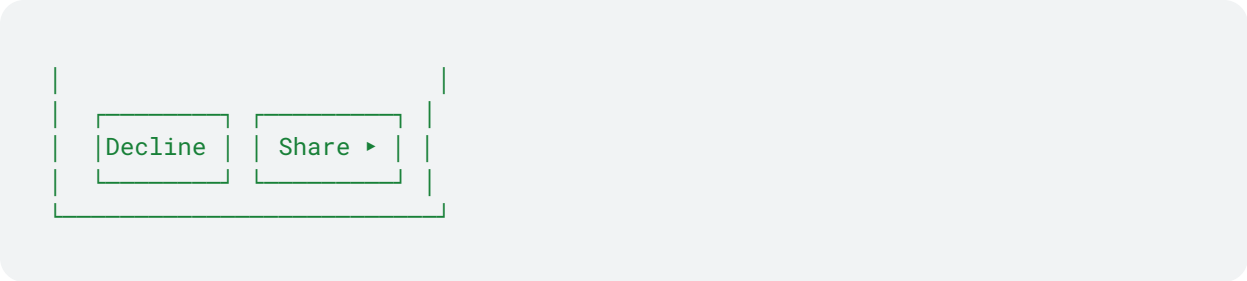
None

```
┌──────────────────────────────┐
|                              |
|  📥 New Credential           |
|                              |
|  An issuer wants to          |
|  issue you a credential:     |
```



8.5 Screen: Present Consent

None



---

9. Credential Format Handling

The frontend must handle credential data dynamically. Here is guidance on how credentials are structured:

9.1 mDoc Credentials

mDoc credentials from the backend will include:[^7][^13][^6]

- `docType`: The document type identifier (e.g., `org.iso.18013.5.1.mDL`, or a Photo ID type)
- `namespaces`: An object where keys are namespace identifiers and values are objects of field name → field value pairs
- `issuerAuth`: The issuer's signature and certificate chain (backend handles validation; frontend just displays metadata)

Frontend rendering rule: Iterate over `namespaces` → for each namespace, iterate over fields → render each as a label (human-readable field name) and value.

9.2 JWT-VC Credentials (if encountered)

JWT-VC credentials will include:

- `type`: Array of credential types
- `credentialSubject`: Object of claims
- `issuer`: DID string
- `issuanceDate`, `expirationDate`: ISO date strings

Frontend rendering rule: Iterate over `credentialSubject` keys → render as label/value pairs.

9.3 Field Type Rendering

Data Type	How to Detect	How to Render
String	`typeof value === 'string'`	Plain text
Number	`typeof value === 'number'`	Formatted number
Boolean	`typeof value === 'boolean'`	"Yes" / "No" or ✓ / ✗
Date (ISO string)	Regex match for ISO date pattern	Formatted date (e.g., "Feb 20, 2026")
Image (base64)	String starts with `data:image/` or field name contains `portrait`, `photo`, `image`	Render as `<img>` tag
Image (URL)	String starts with `https://` and field name suggests image	Render as `<img>` tag
Nested Object	`typeof value === 'object'`	Recursively render key-value pairs, indented
Array	`Array.isArray(value)`	Render as comma-separated list or bulleted list

---

10. Error Handling

Scenario	User-Facing Behavior
Invalid API key	"Authentication failed. Please check your API key and try again."
Token expired (401)	Auto-show re-authentication modal. "Your session has expired. Please enter your API key to continue."

Scenario	User-Facing Behavior
Network error	"Unable to connect to the wallet server. Please check your network and try again."
Invalid offer/request URI	"The provided URI is not recognized. Please check the format and try again."
Credential receive failed	"Failed to receive credential: [backend error message]. Please contact the issuer."
Presentation failed	"Presentation failed: [backend error message]. The verifier could not verify your credential."
No matching credentials for VP request	"You don't have any credentials that match this request."
Camera permission denied	"Camera access is needed for QR scanning. Please enable camera permission in your browser settings, or paste the URI manually below."

---

11. Credential Lifecycle Status Display

Credentials may have different statuses that the backend tracks via Token Status Lists. The frontend should visually indicate status:[^9]

Status	Visual Indicator	Behavior
Active	Green dot or badge	Normal — can be presented
Suspended	Yellow/orange dot or badge, "Suspended" label	Cannot be presented; show a message explaining it's temporarily inactive
Revoked	Red dot or badge, "Revoked" label	Cannot be presented; show a message explaining it's permanently invalid
Expired	Gray dot or badge, "Expired" label	Cannot be presented; show expiration date

---

12. Interoperability Considerations

The wallet is part of a broader PoC that aims for interoperability with external wallets and ecosystems:[^10]

- **Google Wallet**: A secondary goal is to demonstrate credential interoperability with Google Wallet. The frontend does not need to implement this directly, but the data formats and protocols used should align with standard mDoc and OpenID4VCI/VP implementations.
- **Diggiatra**: Another interoperability target. Same principle — standards compliance in the frontend ensures readiness.
- **HAIP (High Assurance Interoperability Profile)**: The backend supports Direct Post with JARM (encrypted responses) for HAIP compliance. The frontend does not need to handle encryption (the backend does), but should be aware that presentation responses may include additional security metadata.[^13][^10]

---

13. Out of Scope

The following are explicitly **not** in scope for this frontend:

- Credential issuance (acting as an issuer) — this frontend is a wallet/holder only
- Verification (acting as a verifier)
- Trust anchor / CA management
- Credential type creation
- Node creation or administration
- Key management (key rotation, key creation)
- DID management (creating sub-DIDs)
- SD-JWT specific handling (not yet stable in the backend)[^10]
- DC API support (not yet implemented in the backend)[^10]
- Duplicate credential detection
- Offline / proximity credential presentation (NFC, BLE)
- Multi-language / i18n (English only for PoC)

---

14. Acceptance Criteria Summary

Authentication

- [ ] Can enter API key and obtain a bearer token
- [ ] Session persists until token expires or tab closes
- [ ] Re-authentication prompt appears on 401

Credential Dashboard

- [ ] All stored credentials appear as stacked cards
- [ ] Cards show credential type and issuer
- [ ] Empty state is handled

Credential Detail

- [ ] Tapping a card expands it with all dynamic fields
- [ ] Fields render correctly for different data types (text, dates, images)
- [ ] Credential status is shown

Receive Credential

- [ ] QR code scanning works on mobile devices
- [ ] Offer URI can be pasted manually
- [ ] Consent screen displays issuer and credential data before acceptance
- [ ] Credential appears in dashboard after acceptance

Present Credential

- [ ] VP request URI triggers the preview flow
- [ ] Consent screen shows verifier identity and requested fields
- [ ] Matched credentials are shown
- [ ] Presentation succeeds and status is displayed

Responsive Design

- [ ] Usable on iPhone/Android at 375px width
- [ ] Usable on tablets at 768px width
- [ ] Usable on desktop at 1440px width
- [ ] All interactive elements have adequate touch targets on mobile

15. Glossary

Term	Definition
SSI	Self-Sovereign Identity — a model where users control their own digital identity
VC	Verifiable Credential — a tamper-evident digital credential signed by an issuer
VP	Verifiable Presentation — a package of one or more VCs presented to a verifier
mDoc	Mobile Document — credential format defined by ISO/IEC 18013-5, CBOR-encoded
mDL	Mobile Driving License — a specific type of mDoc
DID	Decentralized Identifier — a URI-based identifier for entities (e.g., `did:web:example.com`)
OpenID4VCI	OpenID for Verifiable Credential Issuance — protocol for receiving credentials
OpenID4VP	OpenID for Verifiable Presentations — protocol for presenting credentials to verifiers
ID Node	Neoke's server component that handles identity operations via an HTTP API
Bearer Token	A short-lived JWT used to authenticate API requests
API Key	A static key used to obtain a bearer token from the ID Node
CA	Certificate Authority — issues X.509 certificates that form the trust chain for mDoc credentials

Term	Definition
Trust Anchor	A CA root certificate that a verifier trusts for validating credential chains
HAIP	High Assurance Interoperability Profile — a security profile requiring encrypted responses
Selective Disclosure	The ability to share only specific fields from a credential, not the entire document
Holder Binding	Cryptographic proof that the presenter of a credential is its legitimate holder
CBOR	Concise Binary Object Representation — the binary encoding format used by mDoc
DCQL	Digital Credentials Query Language — used by verifiers to specify what they need
Token Status List	A mechanism for tracking credential revocation/suspension without correlation
COSE	CBOR Object Signing and Encryption — the signing mechanism used in mDoc
SPA	Single Page Application — a web app that loads once and updates dynamically

---

References

1. EUDI Wallets with OpenID4VC Protocol Guide - Learn how EUDI Wallets leverage OpenID4VCI issuance and OpenID4VP presentation protocols to enable s...

2. What is OpenID4VCI? The Developer's Guide (2026) - A complete guide to the OpenID4VCI protocol. Understand the full issuance flow, its relation to Open...

3. B2B-Cloud-wallet.pdf - Product/ Flow details The wallet itself has these functions: 1. Receive and store credentials ○...

4. B2B-Cloud-wallet.md - ## Product/ Flow details

The **wallet** itself has these functions:

1. Receive and store credentia...

2. Self-Sovereign Identity: The Ultimate Guide 2025 - Self-Sovereign Identity (SSI) is a model that gives individuals full ownership and control of their ...

3. How Verifiable Credentials Differ from ISO/IEC 18013-5 Ones - The mdoc format, detailed in ISO/IEC 18013-5, defines the structure and specifications for mobile dr...

4. Getting to Know the Verifiable Digital Credential Ecosystem - The mdoc credential format was originally developed to support government-issued mobile IDs, such as...

5. EUDI Wallet Credential Format Guide - ISO/IEC 18013-5:2021: This standard defines the interface specifications for mDLs used in in-person ...

6. Neoke-Platform-mDOC.-Quick-Guide.md - # Neoke IDN Platform — mDoc Photo ID quick guide

Platform: [https://idn.neoke.com](https://idn....

10. ID-node-intro-by-Estrella.docx - --- title: "[]{#_o4wordkog6o8 .anchor}📝 Las notas"

---

18 feb 2026

Estrella - Sebas

Invitados ...

11. OpenID for Verifiable Presentations 1.0 - When using pre-registered Response URIs, the Wallet MUST comply with best practices for redirect URI...

12. SSI wallet for education and research - In the SSI model, the data source, for example an educational institution, gives data or attributes....

13. Issue ISO 18013-5 Mobile Driving License as mDoc - Before you can create the credential template for mDL issuance, you need to set up an issuer root x5...